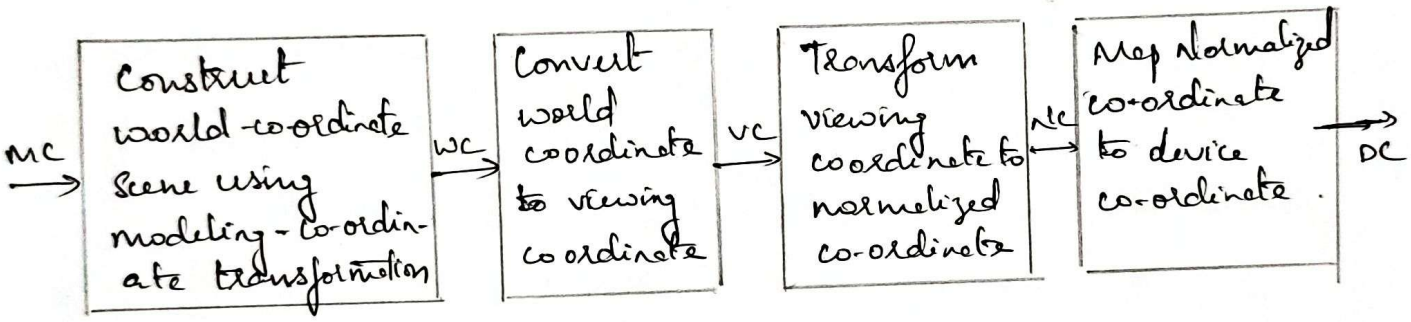
R. Brindha
1BY20CS141
6th sem 'B' sec.

CGV Assignment

1) Build a 2D Viewing transformation pipeline and also explain openGL 2D Viewing functions.

→ • The mapping of a two-dimensional world-coordinate scene description to device coordinates is called a 2D Viewing transformation.
• This transformation is simply referred to as the window to viewport transformation
• We can describe the steps for 2D viewing as indicated



2D Viewing functions:

1) glMatrixMode(mode):
This function sets the current matrix mode for subsequent matrix operations. In 2D viewing, we use GL-MODEL-VIEW and 'GL-PROJECTION' matrix modes.

2) glLoadIdentity():
This function replaces the current matrix with the identity matrix. It is used to reset the transformations before applying new ones.

3) gluOrtho2D(left, right, bottom, top):
This function is a convenience function provided by GLU that sets up a 2D orthographic projection matrix similar to 'glOrtho()'.

7) Build Phong Lighting Model with equation

→ An empirical model for calculating the specular reflection range, developed by Phong Bui Tuong and called the Phong specular-reflection model or simply the Phong model, sets the intensity of specular reflection proportional to $\cos^{n_s}\phi$

→ Angle ϕ can be assigned values in range 0° to 90°.

→ The value assigned to the specular-reflection exponent n_s is determined by the type of surface that we want to display

→ $w(\theta)$ tends to increase as the angle of incidence increases.

→ At $\theta = 90°$, all the incident light is reflected ($w(\theta) = 1$)

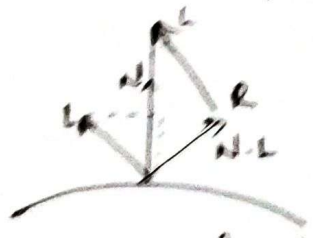
→ Using spectral-reflection function $w(\theta)$, Phong specular-reflection model as

$$I_{l,spec} = w(\theta) I_l \cos^{n_s}\phi$$

I_l is the intensity of light source and ϕ is the viewing angle relative to the specular-reflection direction R

Because V and R are unit vectors in the viewing and specular-reflection directions, we can calculate the value of $\cos\phi$ with the dot product $V \cdot R$.

$$I_{l,spec} = \begin{cases} k_s I_l (V \cdot R)^{n_s}, & \text{if } V \cdot R > 0 \text{ and } N \cdot L > 0 \\ 0.0, & \text{if } V \cdot R \leq 0 \text{ and } N \cdot L \leq 0 \end{cases}$$



- The projection of L onto the direction of the normal vector has a magnitude equal to the dot product N.L, which is also equal to the magnitude of the projection of unit vector R onto the direction of N.
- Therefore, from this diagram, we see that

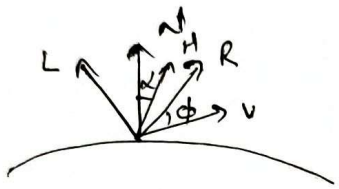
$$R + L = (2N \cdot L) N$$

and the specular-reflection vector is obtained as

$$R = (2N \cdot L)N - L$$

A simplified Phong model is obtained using the halfway vector H between L and V to calculate the range of specular reflections.

If we replace V.R in the Phong model with dot product N.H, this simply replaces the empirical cosφ calculation with empirical cosα calculation



The halfway vector is obtained as

$$H = \frac{L+V}{|L+V|}$$

3) Apply homogeneous co-ordinates for translation, rotation and scaling via matrix representation.

Ans → Translation:

A translation moves all the points in an object along the same straight-line path to new positions.

We can write the components:

$$P'_x = P_x + t_x$$

$$P'_y = P_y + t_y$$

In matrix form:

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Rotation:

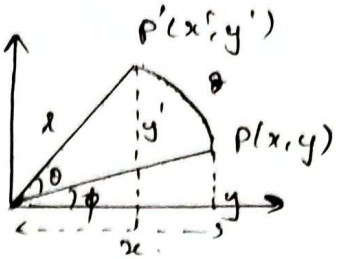
A rotation repositions all points in a object along a circular path in the plane centered at the pivot point in geometry

$\cos\phi = x/r$. $\sin\phi = y/r$

$x = r\cos\phi$ $\quad y = r\sin\phi$

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta - y\cos\theta$



We can write the components

$P'_x = P_x\cos\theta - P_y\sin\theta$

$P'_y = P_x\sin\theta - P_y\cos\theta$

In matrix form

$P' = R \cdot P$ where $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

Scaling:

Scaling changes the size of an object and involves two scale factors s. s_x and s_y for x and y co-ordinates respectively.

Components are

$P'_x = S_x P_x$ and $P'_y = S_y P_y$

in matrix $P' = S \cdot P$ where $S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

Translation $P' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Rotation $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Scaling $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Combining above equation, we can say that

$P' = M_1 * P + M_2$

Co-ordinate (x, y) with homogeneous co-ordinate (x_h, y_h, h) where $x = x_h/h$ and $y = y_h/h$: set $h = 1 \Rightarrow (x, y, 1)$

Homogeneous co-ordinate representation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \longrightarrow \text{translation}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \longrightarrow \text{scaling}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \longrightarrow \text{rotation}$$

4) Outline the differences between raster scan displays and random scan displays.

A

Base of Difference	Raster Scan System	Random Scan System
Electron Beam	The electron beam is swept across the screen, one row at a time, from top to bottom	The electron beam is directed only to the parts of screen where a picture is to be drawn
Resolution	Its resolution is poor because raster system in contrast produces ~~zig~~ zigzag lines	Its resolution is good because this system produces smooth lines drawings.
Picture Definition	Stored as a set of intensity values for all screen points	Stored as a set of line drawing instructions in a display file.
Realistic Display	The capability of this system to store intensity values for pixel makes it well suited for the realistic display	These systems are designed for line drawing and can't display realistic shaded scenes.

5) Demonstrate OpenGL functions for displaying Window Management using GLUT.

A Step 1: Initialization of GLUT:-

* We are using the openGL utility Toolkit.
* We perform the GLUT initialization with the statement

```
glutInit(&argc, argv);
```

* Next, we can state that a display window is to be created on the screen with a given caption for the title bar. This is accomplished with the function.
* `glutCreateWindow ("An example OpenGL prg");`

Here, the single argument for the function can be any character string that we want to use for the display.

* The following function call passes the line-segment description to the display window

```
glutDisplayFunc(lineSegment);
```

* `glutMainLoop();`

This function must be the last one in our program. It displays the initial graphic and puts the program into an infinite loop.

* `glutInitWindowPosition (50, 100);`

The statement specifies that upper-left corner of display window should be placed 50 pixels to the right of left edge of screen and 100 pixels down from top edge of screen

* `glutInitWindowSize (400, 300);`

This function is used to set the initial pixel with width and height of display window.

* `glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);`

The command specifies that a single Refresh is to be used for window and color mode like Red, green and blue component to select color values.

6) Explain the openGL visibility detection function.

A

a) OpenGL polygon-culling functions.

Back-face removal is accomplished with the functions

glEnable (GL_CULL_FACE);

glCullFace (mode)

where parameter mode is assigned the value GL_BACK, GLFRONT, GL_FRONT_AND_BACK

By default, parameter mode is assigned ~~the value~~ GL_BACK, glCullface function has the value GL_BACK

* The culling routine is turned off with

glDisable (GL_CULL_FACE);

b) OpenGL Depth-Buffer functions:-

To use the openGL depth-buffer visibility-detection function. We first need to modify the GL utility Toolkit initialization function for the display mode to include a request for the depth buffer, as well as for refresh buffers.

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

- Depth buffer values can be initialized with

glClear (GL_DEPTH_BUFFER_BIT)

- These routines are activated with the following functions

glEnable (GL_DEPTH_TEST);

- We can set the status of the depth buffer so that it is in a read only state or in a read write state

glDepthMask (write status);

c) OpenGL wire-frame surface visibility method:

A wireframe displays of a standard graphical object can be obtained in OpenGL by requesting that only the edges are to be generated

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)

d) OpenGL - Depth-cuing function

→ We can vary the brightness of an object as a function of its distance from the viewing position with

glEnable (GL_FOG)
glFogi(GL_FOG_MODE, GL_LINEAR)

7) Write the special cases that we discussed with respect to perspective projection transformation

A

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

Special Cases:

1. $z_{prp} = y_{prp} = 0$

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \quad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) \longrightarrow (1)$$

We get (1) when the projection reference point is limited to positions along the zview axis.

2. $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$

$$x_p = x\left(\frac{z_{vp}}{z}\right)$$

$$y_p = y\left(\frac{z_{vp}}{z}\right) \longrightarrow (2)$$

We get (2) when the projection reference point is fixed at co-ordinate origin

3. $z_{vp} = 0$

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right) \longrightarrow (3)\,a$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{pp}\left(\frac{z}{z_{prp} - z}\right) \longrightarrow (3)\,b$$

We get 3a and 3b if the view plane is the uv plane and there is no restrictions on the placement of the projection reference point.

4. $x_{prp} = y_{prp} = z_{prp} = 0$

$$x_p = x\left[\frac{z_{prp}}{z_{prp} - z}\right]$$

$$y_p = y\left[\frac{z_{prp}}{z_{prp} - z}\right]$$

We get 4 with the uv plane as the view plane and the projection references point on the z view axis.

---

8) Explain Bezier curve equation along with its properties

A → Developed by French Engineer Pierre Bezier for use in design of Renault automobile bodies.

→ Bezier have a number of properties that make them highly useful for curve and surface design they are also easy to implement.

→ Bezier curve section can be filled to any number of control points.

Equation

$P_k = (x_k, y_k, z_k)$ P_k - General $(n+1)$ control point positions

P_u - the position vector which describes the path of an approximate Bezier poly polynomial function between P_0 and P_n

$$P(u) = \sum_{k=0}^{n} P_k \, BEZ_{k,n}(u) \qquad 0 \le u \le 1$$

$BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k}$ is the Bernstein polynomial

where $C(n,k) = \dfrac{n!}{k!(n-k)!}$

Properties:

* Basic functions are real
* Degree of polynomial defining the curve is one less than number of defining points.
* Curve generally follows the shape of defining polygon
* Curve connects the first and last control points thus $P(0) = P_0$ and $P(1) = P_n$
* Curve lies within the convex hull of the control points.

---

9) Explain normalization transformation for an Orthogonal projection

A The normalization transformation, we assume that the orthogonal projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame. Also, z-co-ordinate positions for the near and far planes are denoted as z_{near} and z_{far} respectively. This position $(x_{min}, y_{min}, z_{near})$ is mapped to the normalized position $(-1, -1, -1)$ and position $(x_{max}, y_{max}, z_{far})$ is mapped to $(1, 1, 1)$

Transforming the rectangular ~~past~~ parallelopiped view volume to a normalized cube is similar to the method for converting the clipping window into the normalized symmetric square.

The normalization transformation for the orthogonal view volume is

$$M_{ortho.norm} = \begin{bmatrix} \frac{2}{x_{wmax}-x_{wmin}} & 0 & 0 & -\frac{x_{wmax}+x_{wmin}}{x_{wmax}-x_{wmin}} \\ 0 & \frac{2}{y_{wmax}-y_{wmin}} & 0 & -\frac{y_{wmax}+y_{wmin}}{y_{wmax}-y_{wmin}} \\ 0 & 0 & \frac{-2}{z_{near}-z_{far}} & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

10) Explain cohen-sutherland line clipping algorithm.

A Every line endpoint in a picture is assigned a 4-digit binary value. called a region code and each bit position is used to indicate whether the point is inside or outside of one of the clipping window boundaries once we have established region codes for all line endpoints, we can quickly determine which line are completely within clip window and which are clearly outside.

1001	1000	1010
0001	0000	0010
0101	0100	0110

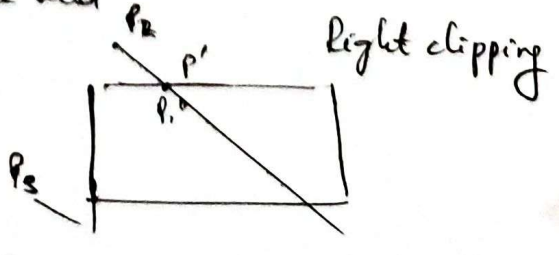
clipping window

When the OR operation between two endpoints region codes for a line segment is false (0000), the line is inside the clipping window.

When AND operation between two endpoints region codes for a line is true, the line is completely outside the clipping window

Lines that cannot be identified as being completely inside or completely outside a clipping window by the region codes tests are next checked for intersection with border lines.

The region codes says P_1 is inside and P_2 is outside



The intersection to be p_1'' and p_2'' is clipped off for the line P_s to P_u we find that point p_s is outside the left boundary and P_u is inside. Therefore, the intersection is P_3 and P_s' is clipped off.

By checking the region codes of P_s' and P_u we find the remainder of the line is below the clipping window and can be eliminated. To determine a boundary intersection point with vertical clipping border line can be obtained by

$$y = y_0 + m(x - x_0)$$

where x is either x_{wmin} or x_{wmax} and slope is

$$m = (y_{end} - y_0) / (x_{end} - x_0)$$

∴ for intersection with horizontal border, the x coordinate is

$$x = y x_0 + \left(\frac{y - y_0}{m} \right).$$